PBS Mapping 2: Developer's Guide

Nicholas M. Boers, Rowan Haigh, and Jon T. Schnute

Fisheries and Oceans Canada
Science Branch, Pacific Region
Pacific Biological Station
3190 Hammond Bay Road
Nanaimo, British Columbia
V9T 6N7

2004

Canadian Technical Report of Fisheries and Aquatic Sciences 2550

Fisheries and Oceans Canada | Pêches et Océans Canada

Canada

Canadian Technical Report of
Fisheries and Aquatic Sciences

Technical reports contain scientific and technical information that contributes to existing knowledge but which is not normally appropriate for primary literature. Technical reports are directed primarily toward a worldwide audience and have an international distribution. No restriction is placed on subject matter and the series reflects the broad interests and policies of the Department of Fisheries and Oceans, namely, fisheries and aquatic sciences.

Technical reports may be cited as full publications. The correct citation appears above the abstract of each report. Each report is abstracted in *Aquatic Sciences and Fisheries Abstracts* and indexed in the Department's annual index to scientific and technical publications.

Numbers 1 - 456 in this series were issued as Technical Reports of the Fisheries Research Board of Canada. Numbers 457 - 714 were issued as Department of the Environment, Fisheries and Marine Service Technical Reports. The current series name was changed with report number 925.

Technical reports are produced regionally but are numbered nationally. Requests for individual reports will be filled by the issuing establishment listed on the front cover and title page. Out-of-stock reports will be supplied for a fee by commercial agents.

Rapport technique canadien des
sciences halieutiques et aquatiques

Les rapports techniques contiennent des renseignements scientifiques et techniques qui constituent une contribution aux connaissances actuelles, mais que ne sont pas normalement appropriés pour la publication dans un journal scientifique. Les rapports techniques sont destinés essentiellement à un public international et ils sont distribués à cet échelon. Il n'y a aucune restriction quant au sujet; de fait, la série reflète la vaste gamme des intérêts et des politiques du ministère des Pêches et des Océans, c'est-à-dire les scences halieutiques et aquatiques.

Les rapports techniques peuvent être cités comme des publications complètes. Le titre exact paraît au-dessus du résumé de chaque rapport. Les rapports techniques sont résumés dans la revue *Résumés des sciences aquatiques et halieutiques*, et ils sont classés dans l'index annual des publications scientifiques et techniques du Ministère.

Les numéros 1 à 456 de cette série ont été publiés à titre de rapports techniques de l'Office des recherches sur les pêcheries du Canada. Les numéros 457 à 714 sont parus à titre de rapports techniques de la Direction générale de la recherche et du développement, Service des pêches et de la mer, ministère de l'Environnement. Les numéros 715 à 924 ont été publiés à titre de rapports techniques du Service des pêches et de la mer, ministère des Pêches et de l'Environnement. Le nom actuel de la série a été établi lors de la parution du numéro 925.

Les rapports techniques sont produits à l'échelon regional, mais numérotés à l'échelon national. Les demandes de rapports seront satisfaites par l'établissement auteur dont le nom figure sur la couverture et la page du titre. Les rapports épuisés seront fournis contre rétribution par des agents commerciaux.

105-01506/

Canadian Technical Report of

Fisheries and Aquatic Sciences 2550

2004

PBS Mapping 2: Developer's Guide

by

Nicholas M. Boers, Rowan Haigh, and Jon T. Schnute

Fisheries and Oceans Canada

Science Branch, Pacific Region

Pacific Biological Station

3190 Hammond Bay Road

Nanaimo, British Columbia

V9T 6N7

CANADA

© Her Majesty the Queen in Right of Canada, 2004

Cat. No. Fs97-6/2550E ISSN 0706-6457

9 8 7 6 5 4 3 2 1 (First printing – September 23, 2004)

Correct citation for this publication:

Boers, N.M., Haigh, R., and Schnute, J.T. 2004. PBS Mapping 2: developer's guide. Can. Tech. Rep. Fish. Aquat. Sci. 2550: 38 + iv p.

TABLE OF CONTENTS

Abstract iv
Résumé iv
1. Introduction 1
2. Software Requirements and Recommendations 2
  2.1. R 3
  2.2. S-PLUS 2000 4
  2.3. S-PLUS 6 4
  2.4. Microsoft Help Workshop 4.03 5
  2.5. Microsoft HTML Help Workshop 5
  2.6. MiKTeX 6
  2.7. MinGW GCC 6
  2.8. Perl 7
  2.9. UNIX Tools 7
  2.10. Emacs 7
3. Directory Structure 10
4. Maintaining the Developer's Version 10
  4.1. Documentation 10
  4.2. Data Sets 11
  4.3. R/S Code 12
  4.4. R/S and C Interface 13
  4.5. C Code 14
  4.6. Testing Changes 15
5. Building Packages for Distribution 15
6. Distributing Packages 17
References 17
Appendix A. PBS Mapping Function Dependencies 19
Appendix B. Bash Shell Scripts 21
  `R.sh` for `PBSmapping` 21
  `SPLUS2000.sh` for `PBSmapping` 23
  `SPLUS6.sh` for `PBSmapping` 26
  `R.sh` for `PBSdata` 30
  `SPLUS2000.sh` for `PBSdata` 31
  `SPLUS6.sh` for `PBSdata` 34
  `PDF.sh` for `PBSmapping` and `PBSdata` 37

LIST OF TABLES

Table 2.1. Required tools for building each type of output package 3
Table 3.1. Directory structure of the PBS software – developer's version 9

LIST OF FIGURES

Figure 1.1. Directory structure for input and output 2

ABSTRACT

Boers, N.M., Haigh, R., and Schnute, J.T. 2004. PBS Mapping 2: developer's guide.
Can. Tech. Rep. Fish. Aquat. Sci. 2550: 38 + iv p.

This report describes the technical process used to develop and build the software package PBS Mapping 2. A companion User's Guide (Schnute et al. 2004) documents the purpose and use of this software, which involves over 10,000 lines of code for R, S-PLUS, and C. To build the package from source files, we use a variety of software tools, most of them freely available from the Internet. We outline procedures for downloading and installing these tools, and we describe the directory tree of source files for building PBS Mapping and a related package PBS Data. We then explain how to use automated scripts that build our libraries for three software environments: R, S-PLUS 2000, and S-PLUS 6. Additional scripts create PDF documentation files for appendices in the User's Guide.

RÉSUMÉ

Boers, N.M., Haigh, R. et Schnute, J.T. 2004. PBS Mapping 2: Guide du développeur.
Can. Tech. Rep. Fish. Aquat. Sci. 2550: 38 + iv p.

Le présent rapport décrit le processus technique qui a été utilisé pour développer et construire le progiciel PBS Mapping 2. Une guide de l'utilisateur (Schnute et al. 2004) décrit par ailleurs l'objet et l'utilisation de ce logiciel, qui comprend plus de 10 000 lignes de codes en R, S-PLUS et C. Pour construire le progiciel à partir des fichiers sources, nous avons utilisé une variété d'outils informatiques, la plupart d'entre eux pouvant être obtenus gratuitement sur Internet. Nous définissons les procédures pour le téléchargement et l'installation de ces outils et nous décrivons la structure des répertoires pour les fichiers sources dans le cadre de la construction de PBS Mapping et un progiciel connexe, PBS Data. Nous expliquons ensuite comment utiliser les scripts automatiques pour mettre en place les librairies pour trois environnements informatiques : R, S-PLUS 2000 et S-PLUS 6. Des scripts supplémentaires créent des fichiers de documentation en format PDF, pour les annexes du Guide de l'utilisateur.

1. INTRODUCTION

The PBS Mapping software package extends the statistical languages R and S-PLUS to include two-dimensional plotting features similar to those commonly available in a Geographic Information System (GIS). Previous reports (Schnute et al. 2003, 2004) provide detailed user manuals for two versions of this software. Over a development period of several years, the underlying computer code has become increasingly complex. This report documents the technical process required to develop and build the PBS Mapping package, along with a supplementary package named PBS Data.

Our work developed from experiences constructing databases that capture information from Canada's Pacific groundfish fisheries. Fishing events take place across a broad range of coastal waters and result in the capture of many species. Initially, we focused on issues related to database design and development, as described in previous reports by Schnute et al. (1996), Haigh and Schnute (1999), Rutherford (1999), Schnute et al. (2001, Section 2 and Appendix A), and Sinclair and Olsen (2002). Analyses of these databases shifted our attention to the problem of portraying and understanding complex spatiotemporal information. Available GIS products are suitable for visual portrayals of geo-referenced data but can be difficult, and sometimes unreliable, for statistical analyses.

At present, we support three platforms for PBS Mapping and PBS Data: R (version 1.90 or higher), S-PLUS 2000, and S-PLUS 6. All of these depend on a single developer's version with

- over 10,000 lines of source code, implemented in R/S-PLUS and C;
- a well-defined directory structure (Figure 1.1), including specific locations for all input (source code, data sets, documentation, etc.) and output (distribution packages);
- a set of tools that facilitate the conversion from input to output;
- a set of scripts that automate the conversion by calling the appropriate tools.

Section 2 of this report outlines the software needed to build PBS Mapping (Version 2), most of it freely available on the Internet. We list sources for each required package and provide suggestions for effective installation. Section 3 details the contents and directory structure of our developer's version. We use scripts to automate the build process from this explicit directory tree. An input directory contains all the raw material (C code, R/S code, R documentation files, etc.), and an output directory receives the packages created, along with various temporary directories and files created during the build process. Section 4 describes the specific objects (documentation, data sets, and code) that a custodian would modify to maintain or enhance the developer's version. Section 5 describes the scripts used to build the the two libraries `PBSmapping` and `PBSdata`, along with documentation for appendices in the User's Guide (Schnute et al. 2004). Finally, Section 6 outlines the structure of the distribution CD and the procedure for submitting the R package to the Comprehensive R Archive Network (CRAN).

PBS Mapping 2 is available on the CRAN website at:
http://cran.r-project.org/src/contrib/Descriptions/PBSmapping.html;
however, this version is specific to R. Users of S will need the distribution CD, available from Jon Schnute (SchnuteJ@pac.dfo-mpo.gc.ca) or Rowan Haigh (HaighR@pac.dfo-mpo.gc.ca).

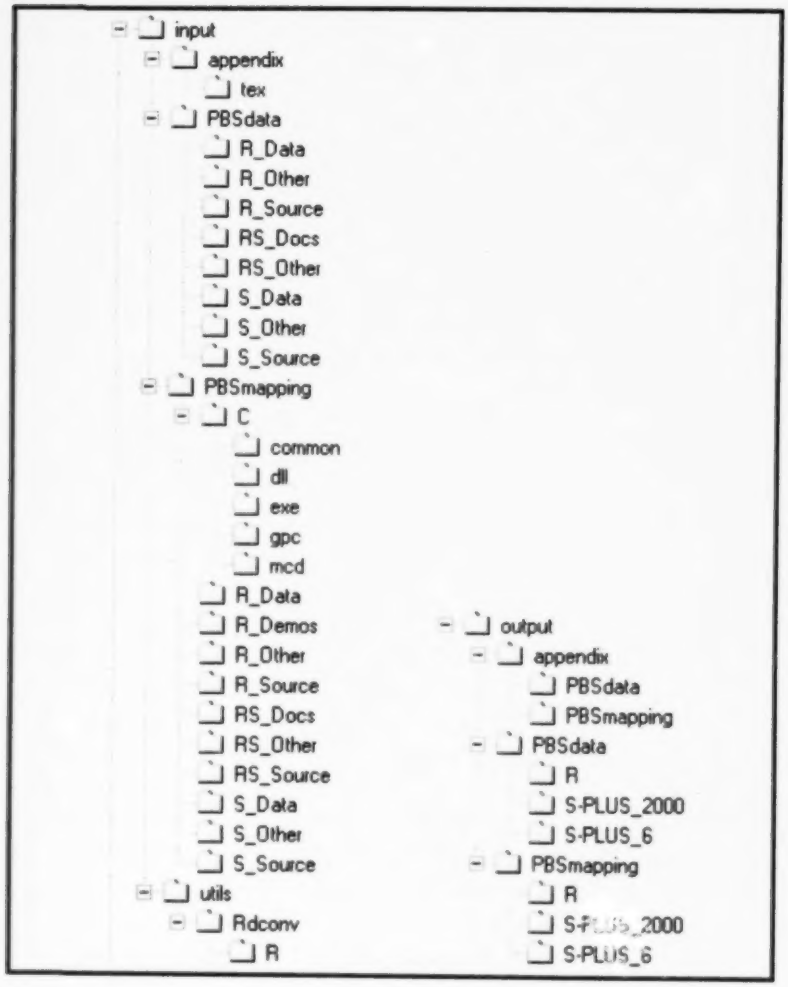

Figure 1. Directory structure for input data and output packages.

2. SOFTWARE REQUIREMENTS AND RECOMMENDATIONS

Building the output files requires a specific set of tools. Table 2.1 lists them and indicates which are required for building each type of package. The subsections that follow introduce briefly each program and its purpose. Furthermore, each description includes installation and post-installation instructions, where necessary. The programs below can be installed in any order; however, we suggest that a developer install them in the order presented here.

Table 2.1. Required tools for building each type of output package. In the table header, R, S2K and S6 refer to R, S-PLUS 2000, and S-PLUS 6 versions, respectively.

Tool	PBS Mapping			PBS Data			Appendix
	R	S2K	S6	R	S2K	S6	
R	✓	✓	✓	✓	✓	✓	✓
S-PLUS 2000	✗	✓	✗	✗	✓	✗	✗
S-PLUS 6	✗	✗	✓	✗	✗	✓	✗
HTML Help Workshop	✓	✗	✓	✓	✗	✓	✗
MiKTeX	✗	✗	✗	✗	✗	✗	✓
MinGW GCC	✓	✓	✓	✗	✗	✗	✗
Perl	✓	✓	✓	✓	✓	✓	✓
UNIX Tools	✓	✓	✓	✓	✓	✓	✓

To run effectively, some software packages require updates to the system's `PATH` environment variable. We indicate the need for such changes in the sections that follow. To implement them in recent versions of Windows (e.g., 2000 and XP):

1. right-click on "My Computer", and select "Properties" from the menu that appears;
2. click on the "Advanced" tab;
3. click on the "Environment Variables" button;
4. in the "System variables" frame, click on the "Path" line followed by the "Edit" button.

To make editing easier, copy the `PATH` string to Notepad, edit the string, and then copy it back. Paths should appear unquoted, even those that contain spaces. Quotes prevent a path from working properly in our scripts, written for a UNIX `bash` shell.

2.1. R

Web site: http://www.r-project.org (The R Project for Statistical Computing)

R is fundamental to building the PBS Mapping and PBS Data packages, as well as typesetting appendices for the User's Guide (Schnute et al. 2004). It includes necessary Perl scripts for creating the distribution packages (R, S-PLUS 2000, and S-PLUS 6). It also includes an extensive conversion system for converting the R documentation files to various formats (Sd, Ssgm, example, html, latex, txt). For S-PLUS 2000, we have modified the R document conversion script, and this stand-alone version appears on our distribution CD as the file `input\utils\Rdconv\Rdconv`.

To begin, perform a standard install of R (version 1.9.0 or later). After installation, ensure its `bin\` directory appears in the system's `PATH` environment variable. Since the Windows operating system is not case sensitive, edit the file `MkRules` in R's `src\gnuwin32\` folder (use a UNIX-aware editor such as GNU Emacs rather than Notepad). Remove the following two sections:

```
.C.d:
     @echo "making $@ from $<"
     @$(CXX) $(DEPARG) $(CXXFLAGS) $($*-CXXFLAGS) $< -o $@
.C.o:
     $(CXX) $(CXXFLAGS) $($*-CXXFLAGS) -c $< -o $@
```

If not removed, these sections cause errors when building the R package. The scripts fail to compile the C code.

To build the technical report's appendices when using R 1.9.1 or earlier, modify the file `Rd.sty` included in R's `share\texmf\` directory. Remove the four lines

```
\newlength{\middle}
\setlength{\middle}{\textwidth}
\addtolength{\middle}{-2.25in}
\addtolength{\middle}{-4pt}
```

since in recent versions of MiKTeX they conflict with the command `\middle`. The R-devel mailing list (<https://stat.ethz.ch/pipermail/r-devel/2004-July/030051.html>) provides further details on the problem.

2.2. S-PLUS 2000

Building PBS Mapping and PBS Data for S-PLUS 2000 requires the commercial software S-PLUS 2000 (Insightful Corporation, http://www.insightful.com/), which converts the source files (`*.S` and `*.dd`) into the binary files that form the package. Section 5 introduces scripts that automate this process. S-PLUS 2000 does not require specific installation options or post-installation configuration.

2.3. S-PLUS 6

Building PBS Mapping and PBS Data for S-PLUS 6 requires the commercial software S-PLUS 6 and its *help file building tools* (Insightful Corporation, http://www.insightful.com/). We use these tools to create a compiled HTML help file (`*.chm`). Then we use the S-PLUS 6 `chapter` command to convert the S-PLUS source files (`*.S` and `*.dd`) into the binary files that form the package. Section 5 introduces scripts that automate this process.

A normal installation does not include the *help file building tools*. During the S-PLUS 6 installation, select "custom install" when prompted for the setup type. The custom install will at some point prompt the user to choose program options. Within the list, expand "Help Files" and then add "Help File Building Tools".

To modify an existing S-PLUS 6 installation, load the Windows "Control Panel" and select "Add or Remove Programs". Find S-PLUS 6 within the program list, select it, and click the "Change" button. Once the S-PLUS 6 setup program begins, choose the "Modify" option and look for "Help File Building Tools".

After installing S-PLUS 6, ensure its `cmd\` directory appears in the system's `PATH` environment variable. Also, find the file `s-doc-general.dtd` in the subdirectory `help\BuildHelpFiles\sgml-tools\dtd\` of the S-PLUS 6 installation tree. Line 31 of this file contains a list of token words

"`code|program|application|file|...|br|verb|p|tag`"

that receive special treatment in the help file compilation. Use a text editor to add `itemize` to this list, so that the line becomes

"`code|program|application|file|...|br|verb|p|tag|itemize`".

Without this change, some of our help files will not compile correctly.

2.4. Microsoft Help Workshop 4.03

Web site: http://www.microsoft.com/downloads/ (Search for "Help Workshop 4.03")

The Microsoft Help Workshop creates standard Windows help files (`*.hlp`) in the Help Compiler 4.0 file format. The available download states that this software is designed for the Windows 95 and 98 operating systems, but it will run under other Microsoft platforms. S-PLUS 2000 expects library help files in this format. The Help Workshop's help file compiler (`hcw.exe`) converts rich-text input (`*.rtf`) into the compiled help output (`*.hlp`). Obtaining rich-text input is the most difficult step in creating these help files. We use several scripts to automate the conversion from R's help file format (`*.Rd`) to rich-text (`*.rtf`). First, the Perl script `Rdconv` converts from Rd to Sd (`nroff`) format. Then, the Perl script `nr2hlp.pl` converts `nroff` files into rich-text format. Section 5 further describes the scripts that automate this process.

After installing the Microsoft Help Workshop, add its directory to the system's `PATH` environment variable.

2.5. Microsoft HTML Help Workshop

Web site: http://msdn.microsoft.com (Search for "HTML Help Workshop")

Microsoft provides the HTML Help Workshop free of charge to create help files for Windows operating systems. This utility includes the HTML help compiler (`hhc.exe`) for converting HTML and graphic files into the binary `.chm` help file format. Recent versions of Windows include a viewer that recognises and displays `.chm` files interactively. Both R and S-PLUS 6 expect help files in this format, and R's Perl scripts and an S-PLUS 6 script (`build_helpfiles`) call the HTML Help Workshop to create them.

After installing the Microsoft HTML Help Workshop, add its directory to the system's `PATH` environment variable.

2.6. MiKTeX

Web site: http://www.miktex.org (TeX distribution)

The program `pdflatex` from the freely available MiKTeX package typesets LaTeX files (`*.tex`) to create PDF files (`*.pdf`). We use this program to create PDFs suitable for appendices in the User's Guide (Schnute et al. 2004). Starting with Rd files (`*.Rd`), the Perl script `Rdconv` creates LaTeX files (`*.tex`). By combining the LaTeX files with either `AppPBSdata.tex` or `AppPBSmapping.tex`, `pdflatex` creates a suitable PDF. Section 5 introduces scripts that automate this process.

Install the "small" version of MiKTeX (currently version 2.4) from the web site, either by downloading a complete binary version (about 24 MB) or running the setup wizard. After installation, `pdflatex` normally defaults to European A4 paper, rather than North American letter size. Change this by using a UNIX-aware editor (such as GNU Emacs) to modify the text file `pdftex.cfg`, which lies in the MiKTeX folder `pdftex\config\`. Explicitly,

- replace `page_width 210 true mm` with `page_width 8.5 true in`
- replace `page_height 297 true mm` with `page_height 11 true in`

Once these changes have been made, run the MiKTeX "Options" program from the "Start" menu. In the "General" options tab, click the button to "Refresh" the database. Also, in the "TeX Formats" tab, select "pdflatex" and click the button to "Build" this format. Finally, ensure that the directory containing `pdflatex.exe` appears in the system's `PATH` environment variable.

The scripts that create R packages require `latex` and use the style file `Rd.sty`, along with other files in the R subdirectory `share\texmf\`. Our scripts routinely copy these files to the temporary working directory where we build the PDF files for documentation purposes.

2.7. MinGW GCC

Web site: http://www.mingw.org/download.shtml (MinGW download page)

The Minimalist GNU for Windows (MinGW) project provides a Windows version of several UNIX utilities. In particular, building packages depends on the GNU Compiler Collection (GCC), binutils, and make. Scripts that will be introduced in Section 5 use these tools to create dynamic-link libraries (DLLs) for R, S-PLUS 2000 and S-PLUS 6. To obtain all three tools, scroll down to the File List table and in the section titled "Current", select the "MinGW" package (`MinGW-3.1.0-1.exe` at the time of writing).

If installing MinGW into "Program Files", name this directory in the installation program as "C:\Progra~1\MinGW\" to avoid spaces. After installing MinGW, open its `bin` directory. In this directory, copy the file `mingw32-make.exe` to `make.exe`. Finally, update the system's `PATH` environment variable to include MinGW's `bin\` directory.

2.8. Perl

Web site: http://www.activestate.com (ActiveState)

The R distribution includes extensive Perl scripts to build packages from source and documentation files. We use them for building all three package types (R, S-PLUS 2000, and S-PLUS 6). The scripts also help convert the R documentation files (`*.Rd`) into a format suitable for each distribution. Navigate to the above website, click on "ActivePerl", then "Download". After submitting a brief registration form, choose the Windows "MSI" package.

When the installation of ActivePerl presents a screen titled "Choose Setup Options", ensure that the option to "Add Perl to the `PATH` environment variable" is checked, as it normally is by default. After installing Perl, its `bin\` directory should appear in the system's `PATH` environment variable. If not, update the `PATH` manually.

2.9. UNIX Tools

Web site: http://sources.redhat.com/cygwin (Cygwin)

UNIX includes a much richer suite of command-line tools than those available in a traditional DOS window. The R utility to build packages and our shell scripts (`*.sh`) to automate the package-building process require a small set of UNIX tools. The easiest way to obtain those tools is through the Cygwin link provided above.

The Cygwin installer `setup.exe` (found at http://sources.redhat.com/cygwin/) provides a GUI that downloads and installs Cygwin components via the Internet. If installing it into "Program Files", enter the directory as "C:\Progra~1\Cygwin\" to avoid spaces. As well, select "DOS" as the "Default Text File Type". The process allows you to choose a download site, and we found reasonable transfer rates from a location called "http://mirrors.kernel.org". Accept all default options, with the following exceptions:

- in the `Archive` section, add `zip`;
- in the `Develop` section, add `flex`;
- in the `Text` section, add `more`

to obtain a suitable Cygwin working environment. To include other packages, select them from the GUI during installation. We purposely do not select the GNU Compiler Collection (GCC) here, so that it does not conflict with the MinGW version. Once the installation completes, add Cygwin's `bin\` folder to the system's `PATH` environment variable, anywhere after MinGW's `bin\` to reduce the chance of conflicts. Additionally, move all `WINNT\` and `WINDOWS\` folders currently specified in the `PATH` to the end to avoid conflicts with Cygwin executables such as `FIND.EXE`.

2.10. Emacs

Web sites: http://www.gnu.org/software/emacs/windows/ntemacs.html (Windows release)
http://ftp.gnu.org/gnu/emacs/windows/ (Windows release downloads)

http://stat.ethz.ch/ESS/ (ESS)
http://www.gnu.org/software/auctex/ (AUCTeX)

Emacs is not required to develop PBS software, but we recommend it. The name Emacs originated as an abbreviation of **E**ditor **mac**ro**s**. Its user manual describes Emacs as an extensible, customisable, self-documenting, real-time display editor for text files (Stallman 2002). It supports a large number of text manipulation and display features, such as

- copying and pasting, including rectangular sections of text;
- automatically filling or wrapping;
- highlighting changes in the current draft; and
- indicating white space at the end of a line.

For a wide variety of computer languages, Emacs also supports language-specific features, such as reserved word recognition, syntax highlighting, and indenting. Extensions can enhance the program's ability to deal with specific applications. For example, Emacs Speaks Statistics (ESS) configures the editor for the R/S programming language (Rossini et al. 2001). Similarly, AUCTeX configures Emacs for the TeX mathematical typesetting system described below. Schnute et al. (2003, Section 5.1 and Appendix B) provide further information about these packages.

Installation requires the two most recent zipped archives for Emacs and ESS. At the time of writing, these are

- `emacs-21.3-fullbin-i386.tar.gz`, available at http://ftp.gnu.org/gnu/emacs/windows/;
- `ess-5.2.2.zip`, available at http://stat.ethz.ch/ESS/downloads/.

Extract all compressed files from the Emacs archive into a suitable folder (e.g., `C:\Program Files\`), making sure that the "Use folder names" option is selected. This creates the subfolder `emacs-21.3\`. Next, extract files from the ESS archive to `C:\Program Files\emacs-21.3\`. This creates the subfolder `ess-5.2.2\`.

After extracting the above files, you need to create the file `C:\.emacs` containing the following two lines:

```
(load "C:/progra~1/emacs-21.3/ess-5.2.2/lisp/ess-site")
(setq ess-fancy-comments nil)
```

Adjust the folder location in the first line to reference the path of your installation. Be sure to place the file `.emacs` in the root directory of drive C, regardless of your Emacs installation location. Also, be sure to use forward slashes in the first line of `.emacs`. (Note: Windows Explorer may not allow you to create `.emacs` or rename a file to `.emacs` because there is no string preceding the dot extension. In this case, create a temporary file `e.txt` with `notepad.exe`, then open a DOS window and type `move c:\e.txt c:\.emacs`.)

You can run Emacs by double-clicking `runemacs.exe` in `C:\Program Files\emacs-21.3\bin\`. Alternatively, if the Emacs `bin\` directory is placed on your PATH, you can issue the command `runemacs` from a DOS window in any directory.

Table 3.1. Directory structure of the developer's version of the PBS software with brief descriptions.

Directory	Description
`input\`	All input files for creating the output files
`  PBSdata\`	PBS Data developer's version
`    RS_Docs\`	Shared documentation (`*.Rd`)
`    RS_Other\`	Shared other files
`    R_Data\`	R data files (`*.rda`)
`    R_Other\`	R other files
`    R_Source\`	R source files (`*.R`)
`    S_Data\`	S data files (`*.dd`)
`    S_Other\`	S other files
`    S_Source\`	S source files (`*.S`)
`  PBSmapping\`	PBS Mapping developer's version
`    C\`	C source code
`      common\`	Code common to DLL & stand-alone executables
`      dll\`	Code specific to the DLL
`      exe\`	Code specific to the stand-alone executables
`      gpc\`	General Polygon Clipper library
`      mcd\`	MemCheckDeluxe
`    RS_Docs\`	Shared documentation (`*.Rd`)
`    RS_Other\`	Shared other files
`    RS_Source\`	Shared source files
`    R_Data\`	R data files (`*.rda`)
`    R_Demos\`	R demo files (`*.R`)
`    R_Other\`	R other files
`    R_Source\`	R source files (`*.R`)
`    S_Data\`	S data files (`*.dd`)
`    S_Other\`	S other files
`    S_Source\`	S source files (`*.S`)
`  appendix\`	Appendix developer's version
`    tex\`	LaTeX source files (`*.tex`)
`  utils\`	Set of utilities used in building the output files
`    Rdconv\`	`Rdconv` script for producing S-PLUS 2000 help
`      R\`	Custom modules used by `Rdconv` script (`*.pm`)
`output\`	All output files created from the input files
`  PBSdata\`	PBS Data distribution packages
`    R\`	R package
`    S-PLUS_2000\`	S-PLUS 2000 library
`    S-PLUS_6\`	S-PLUS 6 library
`  PBSmapping\`	PBS Mapping distribution packages
`    R\`	R package
`    S-PLUS_2000\`	S-PLUS 2000 library
`    S-PLUS_6\`	S-PLUS 6 library
`  appendix\`	Appendix output
`    PBSdata\`	PBS Data appendix
`    PBSmapping\`	PBS Mapping appendix

3. DIRECTORY STRUCTURE

The developer's version of the PBS software contains two directories: `input\` and `output\`. The `input\` directory contains the source files required to build the PBS Mapping package, the PBS Data package, and the technical report's appendix. The `output\` directory contains the resulting packages. Table 3.1 lists the directory structure with further details.

4. MAINTAINING THE DEVELOPER'S VERSION

Users typically see only the R and S-PLUS user interface (documentation, data sets, and R/S functions) to PBS Mapping. Although these three components play a fundamental role, developers must also understand that C code implements many underlying algorithms. Most input files fall into one of five categories:

1. documentation for user-accessible R/S functions and data;
2. data sets for analysis and plotting;
3. high-level R/S code for user-accessible functions in the command-line interface;
4. C code with the purpose of interfacing high-level R/S with low-level C;
5. low-level C code for implementing specific algorithms.

The following subsections provide further details on each component of the software.

4.1. Documentation

The R package system requires documentation for a package's objects (i.e., data sets and functions). It defines a file format (`*.Rd`) for this documentation, where one file generally describes one data set or function (Anonymous 2004). PBS Mapping and PBS Data each contain such documentation, which scripts can convert to various other help file formats for R, S-PLUS 2000, and S-PLUS 6.0, as well as detailed appendices for the User's Guide (Schnute et al. 2004).

Each R documentation file contains several independent sections (e.g., usage, details, examples, etc.). The examples section has two primary purposes. First, it aids users by showing an actual invocation of each function. Second, it provides simple tests of each function, since every time the R scripts rebuild the package they execute all of the examples. If any example fails, the R scripts terminate with an error and identify the function that failed.

The documentation file for each data set includes a `\usage{data(...)}` section that describes how to load the data set in R. The script `remusage.sh`, used when building S-PLUS help files and appendices, removes the usage section for data sets (but not functions) since it only applies to R.

Several issues complicate the conversion from R documentation to S-PLUS 2000 help files. The R documentation format allows hyperlinks. In S-PLUS 2000, hyperlinks referencing a function described in a non-PBS Mapping help file always fail (e.g., links to `par`). To avoid confusion, we remove links to documentation outside of PBS Mapping. Another resolved issue pertains to bullet lists. Previously, the scripts (`Rdconv` and `nr2hlp.pl`) failed to correctly

produce bullet lists in the S-PLUS 2000 help. After modifying both scripts, bullet lists now work. The modified scripts appear in their own directory (`input\utils\Rdconv\`) to make them independent of any particular R version.

4.2. Data Sets

The developer's versions of PBS Mapping and PBS Data include data sets in two formats (`*.dd`, `*.rda`). The former is native to S-PLUS and the latter to R. The formats are generally incompatible, but R can load S-PLUS data dumps (`*.dd`) using the function `data.restore` found in the `foreign` package.

R's `data.restore` function makes transferring data from S-PLUS to R simple. Transferring data in the other direction can prove challenging. Both languages include a `dump` function that writes objects to a file in the format expected by the `source` function. However, using `source` to import a data set is slow, and the current R implementation of `dump` is limited to simple vectors and lists. The R/S functions `write.table` and `read.table` provide an alternative for matrices and data frames, as they efficiently write and read ASCII files. Unfortunately, they do not maintain attributes such as `projection` and `zone`. Given the complications in moving data from R to S-PLUS, it is generally best to manipulate data in S-PLUS, and then import them into R (rather than the opposite).

When using the S-PLUS `data.dump` function, be conscious of the S-PLUS version. In S-PLUS 6, Insightful introduced an updated format for data dumps. Unfortunately, R cannot read this new format. As a solution, set the `oldStyle` argument of the S-PLUS 6 `data.dump` function to `TRUE` when moving data to S-PLUS 2000 or R.

When adding a new data set to either package, only one data set should reside in each dump file. This convention ensures that updating a single data set involves only one file and adding a data set does not involve editing any of the underlying scripts. When the various scripts create a package, they create a list of its data sets based on data set filenames. If two data sets occur in one file, one of them will not be recognized.

Before including a data set based on a PBS Mapping object type (e.g., PolySet), check it for errors. PBS Mapping includes a basic validation routine for each type (i.e., `.validateEventData`, `.validateLocationSet`, `.validatePolyData`, `.validatePolySet`). Additionally, run the `summary` function on it to check its attributes.

Where applicable, new data sets should include `projection` and `zone` attributes. By convention, all EventData and PolySet objects must include the `projection` attribute. When an object spans only one UTM zone, it should include a `zone` attribute.

A complete package requires documentation for each included data set. To ensure maximum portability, write all the documentation in the R documentation format (Section 4.1).

4.3. R/S Code

PBS Mapping makes extensive use of *hidden* functions. Consistent with hidden files in UNIX, their names begin with a period (e.g., `.initPlotRegion`) and are sometimes called *dot functions*. They help avoid code duplication because several higher-level functions can call a single hidden function to perform a specialized task. Conveniently, the R scripts do not require formal documentation files (`*.Rd`) for hidden objects. Appendix A lists dependencies among all PBS Mapping functions, hidden or not.

PBS Mapping is compatible with both R and S-PLUS. We use three methods to achieve this compatibility:

1. including a run-time switch within the R/S code, based on version information;
2. using different source files for R and S-PLUS;
3. modifying the shared code depending on the environment.

Some language-specific issues lend themselves to one method rather than another.

The R/S environment has a list variable `version` with a `language` element that evaluates to the string `"R"` in R and has a `NULL` value in S-PLUS. When code must determine the language environment (R or S-PLUS) at run-time, it detects it by checking the value of `version$language`. We use this method particularly to resolve language-dependent issues that occur infrequently. For example, each language includes its own function to split a string into substrings: `strsplit` in R and `unpaste` in S-PLUS.

We use separate source files in two instances. First, R includes a native `contourLines` function, but S-PLUS does not. The S-PLUS version of PBS Mapping includes a `contourLines` function that imitates the R function. Rather than include this S function in the shared source code, it resides in its own file `contourLines.S`. Second, convention dictates that the `.First.lib` function appears in a file named `zzz.R` (or `zzz.S`). In PBS Mapping, this function displays the name of the software and version, along with a disclaimer of warranty, and then loads the DLL. The R and S-PLUS `.First.lib` functions vary, so each distribution contains its own `zzz` file.

Separate source files also exist to support R's `demo` function that demonstrates a package's features. S-PLUS lacks a `demo` function, so an R-specific demo directory contains all the demo source code. When updating or adding new demos, update the associated file `00Index`.

Finally, we modify shared code for broad changes, equivalent to a search and replace operation. For example, the R `.C` function accepts a `PACKAGE` argument that specifies which DLL contains the C function. The argument always contains the same value, as PBS Mapping uses only one DLL. Therefore, a simple search and replace adds `PACKAGE = "PBSmapping"` to each `.C` call in the R source code. Another difference occurs in S-PLUS 6, which has deprecated our chosen method of using classes. Here, all calls to the `class` function must be replaced with a call to `oldClass`. In our scripts, the UNIX `sed` command makes this type of change.

4.4. R/S and C Interface

Functions in R/S that depend on C functions typically involve three blocks of code that

1. prepare the input data and output memory for the C function;
2. call the C function;
3. structure the output data for the R/S environment.

A detailed description of each block follows.

In PBS Mapping, the arguments to R/S-accessible C functions are always pointers to one-dimensional arrays. Some of a function's arguments correspond to input and others to output. As preparation for the input arguments, R/S code flattens lists, data frames, and other high-level data types into vectors, represented in C as one-dimensional arrays. The R/S code similarly prepares vectors for receiving the C-function output. PBS Mapping uses R/S code to pre-allocate all output memory before entering the C code. Consequently, R/S must estimate the maximum size of the output. A more efficient solution would have C allocate memory for its own output. However, this solution would prevent the same DLL from working with both R and S-PLUS.

In R/S code, the call to a C function names the function being called and lists its arguments. Within the C code, function names generally match the R/S functions that call them. The R/S code explicitly converts all of the arguments to the type expected by the C function (integer, double, etc.). For example, if PBS Mapping passes a PolySet to a C function, the arguments typically include suggestive names like `inID`, `inXY`, `inRows`, `outID`, `outXY`, `outRows`, and `outStatus`. The integer variable `inRows` contains the number of rows in the input PolySet. Similarly, `outRows` reports the number of rows reserved for the output data. The numeric arrays `inID` and `outID` contain all `PID`, `SID`, and `POS` fields in the input and output. Similarly, the arrays `inXY` and `outXY` hold all `X` and `Y` fields in the input and output. A PBS Mapping convention uses `outStatus` to report whether or not the C function was successful.

After a C function terminates, the R/S code first uses the variable `outStatus` to verify the function's success. Possible errors include insufficient physical memory (meaning that the C code tried to allocate memory for internal uses, but failed) and insufficient memory allocated for output (meaning that the estimated output size was not large enough). Depending on the function, other errors are also possible. After a successful call to C, the R/S function proceeds to split output arrays into data frames and other high-level R/S objects.

The file `PBSmapping.c` contains the C code interface, and most of its functions are similar. Each one typically begins with declarations that improve access to the large input and output arrays. For example, if the one-dimensional input array `inID` contains a data frame's `PID`, `SID` and `POS` values, the C function declares corresponding index variables `inPID`, `inSID`, and `inPOS`, and points them to the appropriate locations in the `inID` array. Following such declarations, each C function generally calculates the start and end indices of each polygon in its input arrays. Finally, the function sequentially processes each polygon in the PolySet,

calling a specific routine to manipulate it. Most of the lowest-level routines appear in the file `common\polygons.c` (Section 4.5).

Each R/S accessible function in `PBSmapping.c` makes use of the `goto` statement. Although the `goto` statement is never necessary and generally unpopular, it simplifies error handling. If an error occurs, the function sets the return value `outStatus` and jumps to the function's end where it cleans up all allocated memory. Without the `goto` statement, the code would require either additional variables or repeated tests.

Along the same lines as the `goto` statement, the C routine `joinPolys` uses the `longjmp` function. The `joinPolys` routine depends on the General Polygon Clipper (GPC) (Murta 2004). Traditionally, this library would return to the operating system after generating an error, but this approach is unacceptable in a DLL. By replacing `exit` with `longjmp` in the library, control now returns to `joinPolys`, which then delivers an appropriate error to the R/S environment.

Programming errors in C code can lead to memory leaks, which occur when a C function allocates memory (via `malloc` or a similar function) but fails to deallocate it (via `free`). The consequences of a memory leak can sometimes go unnoticed until a program runs for an extended period. Each R/S-accessible C function includes the lines

```
#ifdef _MCD_CHECK
  showMemStats();
#endif /* _MCD_CHECK */
```

to invoke the MemCheckDeluxe utility in a specially-compiled DLL. This utility's source code, when compiled as part of the DLL for R, will print memory usage statistics on each call to `showMemStats`.

A modified version of MemCheckDeluxe code uses `Rprintf` to print error messages in an R console window. This feature requires linking the `R` library (`libR.a` in R's `src/gnuwin32` directory) into the compiled DLL for R. Further modifications of the code to check memory can make it work in S-PLUS.

4.5. C Code

The low-level PBS Mapping C code (e.g., polygon clip routines) resides primarily in `common\polygons.c`. Interfaces to the low-level functions vary, and they generally require only a minimal set of arguments. We use these programs to make both the PBS Mapping DLL and other stand-alone utilities.

The original C code lies in several directories (e.g., `common`, `dll`, `exe`, `gpc`, `mcd`) to facilitate code development and management. By contrast, the process of building a package compiles this code from a single working directory. Consequently, the corresponding `bash` script first copies all relevant C code into the working directory and then uses a `sed` substitution command to update paths in the `#include` preprocessing directives.

4.6. Testing Changes

When developing PBS Mapping, it is inefficient to create a distribution package before thoroughly testing changes. In an R/S session, a developer can use simple commands to load all of the current code (R/S source and binary DLL). For example, a command similar to

```
source("C:/PBS/input/PBSmapping/RS_Source/PBSmapping")
```

will load the latest R/S source code into an R or S-PLUS session. Then, a command similar to

```
dyn.load("C:/PBS/input/PBSmapping/C/PBSmapping.dll")
```

will load `PBSmapping.dll` into an R session, and a command similar to

```
dll.load("C:/PBS/input/PBSmapping/C/PBSmapping.dll",
         symbols=c("calcOrientation", "clip", "rollupPolys",
                   "calcArea", "calcCentroid", "calcConvexHull",
                   "closePolys", "convUL", "findPolys",
                   "isConvex", "isIntersecting", "joinPolys",
                   "thickenPolys", "thinPolys"));
```

will load `PBSmapping.dll` into an S-PLUS session.

Issuing the command

```
grep "\.C(" PBSmapping | sed "s/.*.C(//g"
```

in `input/PBSmapping/RS_Source` helps produce the list of symbols. To unload a DLL in R, issue a command similar to

```
dyn.unload("C:/PBS/input/PBSmapping/C/PBSmapping.dll")
```

For S-PLUS, a command similar to

```
dll.unload("C:/PBS/input/PBSmapping/C/PBSmapping.dll")
```

will unload the DLL.

5. BUILDING PACKAGES FOR DISTRIBUTION

We distribute PBS Mapping and PBS Data in three formats: R, S-PLUS 2000, and S-PLUS 6. Building each format depends on a unique set of tools and a lengthy sequence of commands (Section 2). Scripts automate these commands, given the appropriate directory structure (Section 3). Before building any of the packages, ensure that your system contains the appropriate software configured according to our suggestions in Section 2.

The developer's `input` directory contains three relevant subdirectories (`PBSmapping\`, `PBSdata\`, `appendix\`). Each of these includes one or more bash shell scripts (`*.sh`) with references to explicit program and directory locations. Before using any script, update it to reflect the correct paths on your computer. Similarly, the commands to build the C code are stored in a Makefile (`input\PBSmapping\C\Makefile`), and you must update its configuration variables, specifically the directory paths for R and Cygwin.

After installing the above software and updating the scripts, building a package is straightforward. In a DOS `cmd` window, change to the appropriate `input\` directory (i.e., `PBSmapping\`, `PBSdata\`, or `appendix\`). The command may resemble

```
cd C:\PBS\input\PBSmapping
```

that navigates to the PBS Mapping development directory. Once in this directory, executing the command

```
bash R.sh YES
```

will begin building an R package. The script `R.sh` includes the important step of running R's checking routine (`RCmd check`). This reports any warnings that occurred while building the package. Be sure to correct any warnings before distributing the package. We provide similar scripts `SPLUS2000.sh` and `SPLUS6.sh` for building S-PLUS 2000 and S-PLUS 6 packages, respectively. Always create an R package before the two S-PLUS versions, since only the R script includes extensive checks for errors. The R build takes longer than the corresponding S builds, partly due to this error checking. Also, the R script builds a source package in addition to a binary package.

To produce a new version of PBS Mapping (e.g., an upgrade from 2.00 to 2.10), update version numbers in the following files:

- `input\PBSmapping\R_Other\DESCRIPTION`,
- `input\PBSmapping\R_Source\zzz.R`,
- `input\PBSmapping\C\DLL\PBSmapping_res.rc`,
- `input\PBSmapping\C\DLL\PBSmapping.c`,
- `input\PBSmapping\RS_Source\PBSmapping`,
- `input\PBSmapping\S_Source\zzz.S`.

Similarly, update version numbers in files for PBS Data to create a new version of that package. Currently, there are only three relevant files:

- `input\PBSdata\R_Other\DESCRIPTION`,
- `input\PBSdata\R_Source\zzz.R`,
- `input\PBSdata\S_Source\zzz.S`.

To make PDF files for appendices in the User's Guide (Schnute et al. 2004), navigate to the directory `input\appendix\` and issue the commands

```
bash PDF.sh PBSmapping
bash PDF.sh PBSdata
```

The output from each build appears in the corresponding output directory. For example, the R script for PBS Mapping creates files in the directory `output\PBSmapping\R\`.

6. DISTRIBUTING PACKAGES

The bash scripts (`*.sh`) introduced in Section 5 create archives with appropriate internal structures (i.e., paths are preserved and files are located in the correct directories). However, the S-PLUS scripts produce files with names that don't follow a standard convention for R (Anonymous 2004). Before adding these S-PLUS archives to the distribution CD or otherwise circulating them, rename them according to the following pattern:

package_version[*_engine*[*_type*]]

where brackets indicate optional components and

- *package* = "PBSmapping" (or "PBSdata");
- *version* = version number;
- *engine* = "R", "Splus2000" or "Splus6";
- *type* = details specific to a particular platform.

For example, if you use the DOS commands

```
cd C:\PBS\input\PBSmapping
bash SPLUS6.sh YES
```

to create the S-PLUS 6 version of PBS Mapping v. 2.00, the script will create a file named `PBSmapping.zip` in `C:\PBS\output\PBSmapping\S-PLUS_6\`. Rename this archive to `PBSmapping_2.00_Splus6.zip`.

After the package passes the `RCmd check` without any warnings (Section 5) and it is appropriately named, submit the R source package to CRAN. Anonymous (2004) provides a complete description of the steps, but a short summary follows:

1. upload the R source distribution (`*.tar.gz`) to ftp://ftp.ci.tuwien.ac.at/incoming;
2. send an e-mail message to cran@r-project.org announcing your upload.

REFERENCES

Anonymous. 2004. Writing R extensions. Version 1.9.1 (June 21, 2004). <http://cran.r-project.org/doc/manuals/R-exts.pdf>.

Haigh, R., and Schnute, J. 1999. A relational database for climatological data. Canadian Manuscript Report of Fisheries and Aquatic Sciences 2472. 26 p.

Murta, A. 2004 Jul 15. General polygon clipper homepage. <http://www.cs.man.ac.uk/aig/staff/alan/software/>.

Rutherford, K.L. 1999. A brief history GFCATCH (1954-1995), the groundfish catch and effort database at the Pacific Biological Station. Canadian Technical Report of Fisheries and Aquatic Sciences 2299. 66 p.

Schnute, J.T., Boers, N.M., and Haigh, R. 2003. PBS software: maps, spatial analysis, and other utilities. Canadian Technical Report of Fisheries and Aquatic Sciences 2496. 82 p.

Schnute, J.T., Boers, N.M., and Haigh, R. 2004. PBS Mapping 2: user's guide. Canadian Technical Report of Fisheries and Aquatic Sciences 2549: 126 p.

Schnute, J.T., Haigh, R., Krishka, B.A., and Starr, P. 2001. Pacific ocean perch assessment for the west coast of Canada in 2001. Canadian Science Advisory Secretariat (CSAS) Research Document 2001/138. 90 p.

Schnute, J.T., Wallace, C.G., and Boxwell, T.A. 1996. A relational database shell for marked Pacific salmonid data (Revision 1). Canadian Technical Report of Fisheries and Aquatic Sciences 2090A. 28 p.

Sinclair, C.A., and Olsen, N. 2002. Groundfish research cruises conducted by the Pacific Biological Station, Fisheries and Oceans Canada, 1944-2002. Canadian Manuscript Report of Fisheries and Aquatic Sciences 2617. 91 p.

Stallman, R. 2002. GNU Emacs manual: fourteenth edition. Free Software Foundation. Boston, MA. 620 p. <http://www.gnu.org/software/emacs/manual/emacs.pdf>.

Rossini, A., Maechler, M., Hornik, K., Heiberger, R.M., Sparapani, R. 2001. Emacs Speaks Statistics: a universal interface for statistical analysis. University of Washington Biostatistics Paper Series, Paper 173. 24 p. <http://www.bepress.com/uwbiostat/paper173>.

APPENDIX A. PBS Mapping Function Dependencies

This appendix documents function dependencies within PBS Mapping. All functions appear as underlined entries in the alphabetic list. If a function depends on others, the list of dependencies appears below the underlined name. Following a standard in UNIX and R, functions whose name begins with a period (*dot functions*) are considered hidden from the user, who would normally use only the non-hidden functions that call them. The names here apply primarily to the R/S working environment, but functions designated '(C)' are implemented in C source code and compiled in the DLL for the mapping package. R/S invokes these functions with the call `.C(...)`.

.addAxis

.addFeature
.addProps
.validatePolyProps

.addLabels

.addProps

.calcDist

.calcOrientation
calcOrientation (C)

.checkProjection

.clip
clip (C)
extractPolyData

.createFastIDdig

.createIDs
.createFastIDdig

.fixGSHHSWorld
findPolys
fixPOS

.initPlotRegion

.rollupPolys
rollupPolys (C)

.plotMaps
.addAxis
.addLabels
.initPlotRegion
.validateXYData
addLines
addPoints
addPolys

.preparePolyProps
.createIDs
.validatePolyData

.validateData
.createIDs

.validateEventData
.validateData

.validateLocationSet
.validateData

.validatePolyData
.validateData

.validatePolyProps
.validateData

.validatePolySet
.validateData

.validateXYData
.validateData

addLabels
.addFeature
.checkProjection
.validateEventData
.validatePolyData
.validatePolySet
calcCentroid
calcMidRange
calcSummary
is.EventData
is.PolyData

addLines
.addProps
.checkProjection
.clip
.createFastIDdig
.createIDs
.preparePolyProps
.validatePolyProps
.validatePolySet
is.PolyData

addPoints
.addFeature
.checkProjection
.validateEventData
.validatePolyData
is.PolyData

addPolys
.addProps
.checkProjection
.clip
.createFastIDdig
.createIDs
.preparePolyProps
.rollupPolys
.validatePolyProps
.validatePolySet
is.PolyData

addStipples
.addFeature
.checkProjection
.clip
.validatePolySet
findPolys
is.PolyData
thickenPolys

appendPolys
.validatePolySet
is.PolySet

as.EventData
.validateEventData
is.EventData

as.LocationSet
.validateLocationSet
is.LocationSet

as.PolyData
.validatePolyData
is.PolyData

as.PolySet
.validatePolySet
is.PolySet

calcArea
.rollupPolys
.validatePolySet
calcArea (C)
convUL
is.PolyData

calcCentroid
.rollupPolys
.validatePolySet
calcCentroid (C)
is.PolyData

calcConvexHull
.calcDist
.createIDs
.rollupPolys
.validatePolySet
.validateXYData
calcConvexHull (C)
is.PolyData
is.PolySet

calcMidRange
.validatePolySet
calcSummary
is.PolyData

calcSummary
.createIDs
.rollupPolys
.validatePolySet
is.PolyData

clipLines
.clip
.validatePolySet
is.PolySet

clipPolys
.clip
.validatePolySet
is.PolySet

closePolys
.validatePolySet
closePolys (C)
is.PolySet

combineEvents
.validateEventData
is.PolyData
contourLines

convCP
is.PolyData
is.PolySet

convDP
.validatePolyData
is.PolySet

convLP
.validatePolySet
is.PolySet

convUL
.validateXYData
convUL **(C)**

extractPolyData
.createIDs
.validatePolySet
is.PolyData

findPolys
.validateEventData
.validatePolySet
findPolys **(C)**
is.LocationSet

fixBound
.validatePolySet
is.PolySet

fixPOS
.rollupPolys
.validatePolySet
is.PolySet

is.EventData
.validateEventData

is.LocationSet
.validateLocationSet

is.PolyData
.validatePolyData

is.PolySet
.validatePolySet

isConvex
.validatePolySet
is.PolyData
isConvex **(C)**

isIntersecting
.validatePolySet
is.PolyData
isIntersecting **(C)**

joinPolys
.validatePolySet
is.PolySet
joinPolys **(C)**

locateEvents
is.EventData

locatePolys
.validatePolyData
is.PolySet

makeGrid
is.PolySet

makeProps
.validatePolyData
is.PolyData

makeTopography

plotLines
.plotMaps
is.PolyData

plotMap
.plotMaps
is.PolyData

plotPoints
.plotMaps
is.PolyData

plotPolys
.plotMaps
is.PolyData

print.EventData
summary.EventData

print.LocationSet
summary.LocationSet

print.PolyData
summary.PolyData

print.PolySet
summary.PolySet

print.summary.PBS

summary.EventData

summary.LocationSet
.createIDs

summary.PolyData
.createIDs

summary.PolySet
.createIDs

thickenPolys
.calcDist
.createIDs
.validatePolySet
is.PolySet
thickenPolys **(C)**

thinPolys
.validatePolySet
is.PolySet
thinPolys **(C)**

APPENDIX B. Bash Shell Scripts

Bash shell scripts for building PBSmapping (`R.sh`, `SPLUS2000.sh`, `SPLUS6.sh`), **PBSdata** (`R.sh`, `SPLUS2000.sh`, `SPLUS6.sh`), **and the User's Guide appendices** (`PDF.sh`).

R.sh for PBSmapping

```
# Cygwin's UNIX tools (sed, grep, cd, mkdir, cp, mv, ...) should be on the path.

#--CONFIGURATION-------------------------------------------------------------

# Name of package
PKG_NAME=PBSmapping

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo
# Input and output directories
IN_DIR=$PROJ_ROOT/input/PBSmapping
OUT_DIR=$PROJ_ROOT/output/PBSmapping/R

#--FUNCTIONS-----------------------------------------------------------------

# Help display
# ------------
displayInstructions() {
    echo '
This script will create the PBS Mapping package for R.  If you are producing
a new version, be sure to update version numbers in the following locations:

  * .\R_Other\DESCRIPTION
  * .\R_Source\zzz.R
  * .\C\DLL\PBSmapping_res.rc

If you have, pass "YES" as an argument to this script.'
    exit 1
}

# Remove existing packages
# ------------------------
cleanUp() {
    # if the output directory exists
    if [ -e $OUT_DIR ]; then
       # delete it
       rm -vr $OUT_DIR
       # check the return value of 'rm'; if it was unsuccessful,
       # exit with an error code
       if [ $? -ne 0 ]; then exit 1; fi
    fi
}

# Build the structure required by the R scripts
# ---------------------------------------------
buildStructure() {
    cd $IN_DIR

    echo 'Creating R output directory*****************************'
    mkdir -v $OUT_DIR

    echo 'Creating package directory*****************************'
    mkdir -v $OUT_DIR/$PKG_NAME
```

```
    cp -v R_Other/DESCRIPTION $OUT_DIR/$PKG_NAME/

    echo 'Creating data directory*******************************'
    mkdir -v $OUT_DIR/$PKG_NAME/data
    cp -v R_Data/*.rda $OUT_DIR/$PKG_NAME/data

    echo 'Creating demo directory*******************************'
    mkdir -v $OUT_DIR/$PKG_NAME/demo
    cp -v R_Demos/00Index $OUT_DIR/$PKG_NAME/demo
    cp -v R_Demos/*.R $OUT_DIR/$PKG_NAME/demo

    echo 'Creating documentation directory**********************************'
    mkdir -v $OUT_DIR/$PKG_NAME/man
    cp -v RS_Docs/*.Rd $OUT_DIR/$PKG_NAME/man
    # remove 'contourLines.Rd', since our contourLines() function only exists
    # in the S-PLUS distribution
    rm -vf $OUT_DIR/$PKG_NAME/man/contourLines.Rd

    echo 'Creating R source code directory**********************************'
    mkdir -v $OUT_DIR/$PKG_NAME/R
    cp -v RS_Source/PBSmapping $OUT_DIR/$PKG_NAME/R/PBSmapping.R
    cp -v R_Source/zzz.R $OUT_DIR/$PKG_NAME/R/
    # Add 'PACKAGE = "PBSmapping"' to the .C() function calls.
    # The S-PLUS .C function does not expect an argument 'PACKAGE'; however,
    # the R .C function does. Instead of having an "if the language is S" 'if'
    # statement in the code, do a simple text substitution here using 'sed'.
    cd $OUT_DIR/$PKG_NAME/R
    sed 's/outStatus = integer(1));/outStatus = integer(1), PACKAGE =
"PBSmapping");/g' PBSmapping.R > tmp
    mv -fv tmp PBSmapping.R
    cd $IN_DIR

    echo 'Creating C source code directory**********************************'
    mkdir -v $OUT_DIR/$PKG_NAME/src
    cp -v C/common/*.c $OUT_DIR/$PKG_NAME/src
    cp -v C/common/*.h $OUT_DIR/$PKG_NAME/src
    cp -v C/gpc/*.c $OUT_DIR/$PKG_NAME/src
    cp -v C/gpc/*.h $OUT_DIR/$PKG_NAME/src
    cp -v C/dll/*.c $OUT_DIR/$PKG_NAME/src
    cp -v C/dll/*.rc $OUT_DIR/$PKG_NAME/src
    cp -v ../utils/runsed.sh $OUT_DIR/$PKG_NAME/src
    cp -v ../utils/dos2unix.exe $OUT_DIR/$PKG_NAME/src
    # flatten C source code: the C code normally falls into several
    # directories; make it so all the code appears in a single directory.

    # Build a file 'sedscr' (below) that is used by the shell script
    # 'runsed.sh'.  The call to 'runsed.sh' causes the program 'sed' to execute
    # the script 'sedscr' on all files that match the wildcard (*.c below).
    cd $OUT_DIR/$PKG_NAME/src
    echo 's/\.\.\/common\///' > sedscr
    echo 's/\.\.\/gpc\///' >> sedscr
    echo 's/\.\.\/mcd\///' >> sedscr
    ./runsed.sh *.c
    # 'runsed.sh' produces backup files (BAK); remove them and the script itself
    rm -vf *.bak runsed.sh sedscr
    # 'sed' output contains DOS line endings; convert them to UNIX
    ./dos2unix *.c
    # Remove 'dos2unix.exe' so that it's not in the final TAR.GZ archive (RH)
    rm -vf dos2unix.exe
    cd $IN_DIR

    echo 'Creating other file directory*************************************'
    mkdir -v $OUT_DIR/$PKG_NAME/inst
```

```
    cp -v RS_Other/COPYING $OUT_DIR/$PKG_NAME/inst/

    # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
    # If adding the tech. report to the PBS Mapping package, update and then
    # uncomment the following line:
    #
    # cp -v RS_Other/*.pdf $OUT_DIR/$PKG_NAME/inst/
    #
    # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
}

# Run the R scripts to build the package; first check the directory,
# and then build source (ZIP) and binary (TAR.GZ) distributions.
# ------------------------------------------------------------------
buildPackage() {
    cd $OUT_DIR
    RCmd CHECK $PKG_NAME
    if [ $? -eq 0 ]
       then
       RCmd BUILD $PKG_NAME
       RCmd BUILD --binary $PKG_NAME

       rm -vr PBSmapping PBSmapping.Rcheck
       if [ $? -ne 0 ]; then exit 1; fi
    fi
    cd ..
}

#--RUN-------------------------------------------------------------------------

if [ "$1" != "YES" ]
    then
    displayInstructions
    else
    cleanUp
    buildStructure
    buildPackage
fi
```

SPLUS2000.sh for PBSmapping

```
# Cygwin's UNIX tools (sed, grep, cd, mkdir, cp, mv, ...) should be on the path.

#--CONFIGURATION---------------------------------------------------------------

# Name of package
PKG_NAME=PBSmapping

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo
# Input and output directories
IN_DIR=$PROJ_ROOT/input/PBSmapping
OUT_DIR=$PROJ_ROOT/output/PBSmapping/S-PLUS_2000

# S-PLUS 2000's 'cmd' directory
SP2000_CMD=D:/Splus/cmd

#--FUNCTIONS-------------------------------------------------------------------

# Help display
# ------------
displayInstructions() {
```

```
    echo '
This script creates the PBS Mapping package for S-PLUS 2000.
If you are producing a new version, be sure to update version numbers
in the following locations:

  * .\S_Source\zzz.S
  * .\C\DLL\PBSmapping_res.rc

Then pass "YES" as an argument to this script.'
    exit 1
}

# Remove existing packages
# ------------------------
cleanUp() {
    if [ -e $OUT_DIR ]; then
       rm -vr $OUT_DIR
       if [ $? -ne 0 ]; then exit 1; fi
    fi
}

# Build help files
# ----------------
buildHelp() {
    cd $IN_DIR

    echo 'Creating help (PBSmapping.hlp)******************************'
    mkdir -v $OUT_DIR
    mkdir -v $OUT_DIR/$PKG_NAME
    mkdir -v $OUT_DIR/$PKG_NAME/nroff
    cp -v RS_Docs/*.Rd $OUT_DIR/$PKG_NAME/nroff/
    cp -v ../utils/remusage.sh $OUT_DIR/$PKG_NAME/nroff/
    cp -v ../utils/nr2hlp.pl $OUT_DIR/$PKG_NAME/nroff
    cp -vr ../utils/Rdconv/* $OUT_DIR/$PKG_NAME/nroff

    cd $OUT_DIR/$PKG_NAME/nroff/

    # Remove \usage section from each .Rd file,
    # since they don't apply to the S-PLUS help file.
    bash ./remusage.sh
    rm -v remusage.sh

    # Convert *.Rd to .d
    for RdFile in *.Rd ; do
       perl Rdconv -t Sd -o `basename $RdFile .Rd`.d $RdFile
    done
    rm -v *.Rd

    # Convert *.d to .rtf
    perl nr2hlp.pl PBSmapping
    mv -v PBSmappi.rtf PBSmapping.rtf

    # Compile .hlp file
    hcw /c /e /m PBSmappi

    cd ..

    # Copy help file and clean up temporary 'nroff' directory
    mv -v nroff/PBSmappi.hlp PBSmapping.hlp
    rm -vrf nroff

    cd $IN_DIR
}
```

```
# Build the DLL
# -------------
buildDLL() {
    cd $IN_DIR

    echo 'Creating DLL (PBSmapping.dll)******************************'
    mkdir -v $OUT_DIR/$PKG_NAME/src
    cp -v C/Makefile $OUT_DIR/$PKG_NAME/src/

    mkdir -v $OUT_DIR/$PKG_NAME/src/common/
    cp -v C/common/*.c $OUT_DIR/$PKG_NAME/src/common/
    cp -v C/common/*.h $OUT_DIR/$PKG_NAME/src/common/

    mkdir -v $OUT_DIR/$PKG_NAME/src/gpc/
    cp -v C/gpc/*.c $OUT_DIR/$PKG_NAME/src/gpc/
    cp -v C/gpc/*.h $OUT_DIR/$PKG_NAME/src/gpc/

    mkdir -v $OUT_DIR/$PKG_NAME/src/dll/
    cp -v C/dll/*.c $OUT_DIR/$PKG_NAME/src/dll/
    cp -v C/dll/*.rc $OUT_DIR/$PKG_NAME/src/dll/

    cd $OUT_DIR/$PKG_NAME/src/
    make splus2000
    mv -v PBSmapping.dll ../
    make clean

    cd $IN_DIR
}

# Build the rest of the package
# -----------------------------
buildPackage() {
    cd $IN_DIR

    echo 'Copying S-PLUS data****************************'
    cp -v S_Data/*.dd $OUT_DIR/$PKG_NAME/

    echo 'Copying S-PLUS source code*********************************'
    cp -v RS_Source/PBSmapping $OUT_DIR/$PKG_NAME/PBSmapping.S
    cp -v S_Source/*.S $OUT_DIR/$PKG_NAME/

    echo 'Copying other files****************************'
    cp -vrf RS_Other/* $OUT_DIR/$PKG_NAME/

    cd $OUT_DIR/$PKG_NAME/

    echo 'Creating library directory structure*****************************'
    mkdir -v _Data
    mkdir -v _Data/_Help
    mkdir -v _Prefs

    # The .First.lib() function in zzz.S is "open",
    # meaning that it is missing its closing brace;
    # append some additional commands to the function and then close it.
    grep '.C(' PBSmapping.S | sed -r s/.+\(\".+\"\).+/\\1,/g | sed -r \$s/,//g >
tmpCFUNCS
    echo 'dll.load (paste(search()[library(PBSmapping)], "\\..\\PBSmapping.dll",
sep=""), c(' >> zzz.S
    cat tmpCFUNCS >> zzz.S
    echo '    ), "cdecl");' >> zzz.S
    echo '}' >> zzz.S
```

```
    # Aadd commands to the file zzz.S to cause S-PLUS to load the data sets
    for i in *.dd
      do echo "data.restore(\"$i\");" >> zzz.S
    done

    # Prepare 'script'; this file will be passed to S-PLUS 2000 in batch mode;
    # it should cause S-PLUS to source all of the source (*.S) files
    echo '
source ("PBSmapping.S");
source ("contourLines.S");
source ("zzz.S");
' > script

    # Start S-PLUS in batch mode using 'script'
    echo 'Building library*****************************'
    $SP2000_CMD/splus /BATCH script S_PROJ=.

    # Finish cleaning up
    echo 'Cleaning up*****************************'
    mv -v *.S src/
    rm -vf *.dd script tmp*
    rm -vrf _Prefs/*

    cd ..

    zip -r -9 $PKG_NAME $PKG_NAME
    rm -vrf $PKG_NAME
}

#--RUN-------------------------------------------------------------------------
if [ "$1" != "YES" ]
    then
    displayInstructions
    else
    cleanUp
    buildHelp
    buildDLL
    buildPackage
fi
```

SPLUS6.sh for PBSmapping

```
# Cygwin's UNIX tools (sed, grep, cd, mkdir, cp, mv, ...) should be on the path.

#--CONFIGURATION---------------------------------------------------------------

# Name of package
PKG_NAME=PBSmapping

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo
# Input and output directories
IN_DIR=$PROJ_ROOT/input/PBSmapping
OUT_DIR=$PROJ_ROOT/output/PBSmapping/S-PLUS_6

# R root directory
R_ROOT=D:/R/rw1091

# Cygwin bin directory
CYG_BIN=D:/Utils/Cygwin/bin

# Microsoft HTML Help Workshop directory
```

```
HHW_DIR=D:/Utils/HTML-Help-Workshop

# S-PLUS 6.x's CHM tools directory
CHM_TOOLS_DIR=D:/Splus62/help/BuildHelpFiles

#-----------------------------------------------------------------------
# Variables for running the S-PLUS 6 'build_helpfiles' script
# See build_helpchm.cyg in $CHM_TOOLS_DIR
#-----------------------------------------------------------------------
CHM_DIR=$OUT_DIR/$PKG_NAME  # Path to CHM file (that will be created)
CHM_NAME=$PKG_NAME          # Basename of the CHM file
HEADER="Language Reference for Library PBS Mapping"
DEF_PAGE=SOM.html           # Default page when help first loaded
SEARCH_STOPIC_LIST=T        # S-topic list to link to splus.chm
STOPIC_LIST_SPLUS=$CHM_TOOLS_DIR/stopicList.out
#GUI_HELP_DIR=$CHM_DIR/html  # If including HTML files
HAVE_HTML_FILES=F
SGML_DIR=$CHM_DIR/sgml      # Path to SGML files
HTML_DIR=$SGML_DIR          # Path to root of HTML files


#--FUNCTIONS-----------------------------------------------------------------

# Help display
# ------------
displayInstructions() {
    echo '
This script creates the PBS Mapping package for S-PLUS 6.
If you are producing a new version, be sure to update version numbers
in the following locations:

  * .\S_Source\zzz.S
  * .\C\DLL\PBSmapping_res.rc

Then pass "YES" as an argument to this script.'
    exit 1
}

# Remove existing packages
# ------------------------
cleanUp() {
    if [ -e $OUT_DIR ]; then
      rm -vr $OUT_DIR
      if [ $? -ne 0 ]; then exit 1; fi
    fi
}

# Build help files
# ----------------
buildHelp() {
    cd $IN_DIR
    echo 'Creating help (PBSmapping.chm)*****************************'
    mkdir -v $OUT_DIR
    mkdir -v $OUT_DIR/$PKG_NAME
    mkdir -v $OUT_DIR/$PKG_NAME/sgml

    cp -v RS_Docs/*.Rd $OUT_DIR/$PKG_NAME/sgml/
    cp -v ../utils/remusage.sh $OUT_DIR/$PKG_NAME/sgml/
    cp -v S_Other/SOM.sgml $OUT_DIR/$PKG_NAME/sgml/
    cp -vr ../utils/Rdconv/* $OUT_DIR/$PKG_NAME/sgml/

    cd $OUT_DIR/$PKG_NAME/sgml/
    # The script 'remusage.sh' will remove the usage section of each
```

```
    # .Rd file, since that section doesn't apply to the S-PLUS 6 help
    bash ./remusage.sh
    rm -vf remusage.sh

    # Convert all the .Rd files to .sgml
    for RdFile in *.Rd ; do
       perl Rdconv -t Ssgm -o `basename $RdFile .Rd`.sgml $RdFile
    done
    rm -v *.Rd

    # call the S-PLUS utility that will build the help files
    $CHM_TOOLS_DIR/build_helpfiles $CHM_NAME $SGML_DIR $HTML_DIR $CHM_TOOLS_DIR
$HHW_DIR $DEF_PAGE $CHM_DIR $CYG_BIN "$HEADER" $SEARCH_STOPIC_LIST $STOPIC_LIST_SPLUS
$GUI_HELP_DIR $HAVE_HTM_FILES

    # Check help.log for errors
    cd ..
    nerr=$(grep -c -i -w "error" help.log)
    echo $nerr
    if [ $nerr -gt 0 ]; then
       echo Terminating: Check help.log \in `pwd | cygpath -ms -f -`
       exit 1
    fi

    mv help.log $OUT_DIR/
    rm -rf sgml
    rm -rf deadlinks_postfix.lst deadlinks_prefix.lst help.log
    cd $IN_DIR
}

# Build the DLL
# -------------
buildDLL() {
    cd $IN_DIR

    echo 'Creating DLL (PBSmapping.dll)******************************'
    mkdir -v $OUT_DIR/$PKG_NAME/src
    cp -v C/Makefile $OUT_DIR/$PKG_NAME/src/

    mkdir -v $OUT_DIR/$PKG_NAME/src/common/
    cp -v C/common/*.c $OUT_DIR/$PKG_NAME/src/common/
    cp -v C/common/*.h $OUT_DIR/$PKG_NAME/src/common/

    mkdir -v $OUT_DIR/$PKG_NAME/src/gpc/
    cp -v C/gpc/*.c $OUT_DIR/$PKG_NAME/src/gpc/
    cp -v C/gpc/*.h $OUT_DIR/$PKG_NAME/src/gpc/

    mkdir -v $OUT_DIR/$PKG_NAME/src/dll/
    cp -v C/dll/*.c $OUT_DIR/$PKG_NAME/src/dll/
    cp -v C/dll/*.rc $OUT_DIR/$PKG_NAME/src/dll/

    cd $OUT_DIR/$PKG_NAME/src/
    make splus6
    mv PBSmapping.dll ../S.dll
    make clean

    cd $IN_DIR
}

# Build the rest of the package
# -----------------------------
buildPackage() {
    cd $IN_DIR
```

```
    echo 'Copying S-PLUS data*****************************'
    cp -v S_Data/*.dd $OUT_DIR/$PKG_NAME/

    echo 'Copying S-PLUS source code*********************************'
    cp -v RS_Source/PBSmapping $OUT_DIR/$PKG_NAME/PBSmapping.S
    cp -v S_Source/*.S $OUT_DIR/$PKG_NAME/

    echo 'Copying other files*****************************'
    cp -vrf RS_Other/* $OUT_DIR/$PKG_NAME/

    cd $OUT_DIR/$PKG_NAME/

    echo 'Updating PBSmapping.S*****************************'
    # S-PLUS deprecated the 'class' function; they renamed it to 'oldClass'
    sed 's/class(/oldClass(/g' PBSmapping.S > tmp
    sed 's/unoldClass(/unclass(/g' tmp > tmp2
    mv -vf tmp2 PBSmapping.S

    echo 'Updating zzz.S*****************************'
    # add additional lines to 'zzz.S' to enable support for our "old" classes
    echo 'setOldClass(c("EventData", "data.frame"));' >> zzz.S
    echo 'setOldClass(c("LocationSet", "data.frame"));' >> zzz.S
    echo 'setOldClass(c("PolyData", "data.frame"));' >> zzz.S
    echo 'setOldClass(c("PolySet", "data.frame"));' >> zzz.S
    echo '}' >> zzz.S
    # add the lines that will source the data files
    for i in *.dd
      do echo "data.restore(\"$i\");" >> zzz.S
    done

    # call the S-PLUS 'CHAPTER' command, which will source all of the *.S files
    # and build the package
    echo 'Building library*****************************'
    CHAPTER -s

    echo 'Cleaning up*****************************'
    mv *.S src/
    rm -rf *.dd .Data/__Hhelp

    cd ..

    zip -r -9 $PKG_NAME $PKG_NAME
    rm -vrf $PKG_NAME
}

#--RUN-------------------------------------------------------------------------
if [ "$1" != "YES" ]
    then
    displayInstructions
    else
    cleanUp
    buildHelp
    buildDLL
    buildPackage
fi
```

R.sh for PBSdata

```
# Cygwin's UNIX tools (sed, grep, cd, mkdir, cp, mv, ...) should be on the path.

#--CONFIGURATION-------------------------------------------------------------

# Name of package
PKG_NAME=PBSdata

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo
# Input and output directories
IN_DIR=$PROJ_ROOT/input/PBSdata
OUT_DIR=$PROJ_ROOT/output/PBSdata/R

#--FUNCTIONS-----------------------------------------------------------------

# Help display
# ------------
displayInstructions() {
    echo '
This script will create the PBS Data package for R.  If you are producing
a new version, be sure to update version numbers in the following locations:

  * .\R_Other\DESCRIPTION
  * .\R_Source\zzz.R

If you have, pass "YES" as an argument to this script.'
    exit 1
}

# Remove existing packages
# ------------------------
cleanUp() {
    if [ -e $OUT_DIR ]; then
       rm -vr $OUT_DIR
       if [ $? -ne 0 ]; then exit 1; fi
    fi
}

# Build the structure required by the R scripts
# ---------------------------------------------
buildStructure() {
    cd $IN_DIR

    echo 'Creating R output directory*******************************'
    mkdir -v $OUT_DIR

    echo 'Creating package directory********************************'
    mkdir -v $OUT_DIR/$PKG_NAME
    cp -v R_Other/DESCRIPTION $OUT_DIR/$PKG_NAME/

    echo 'Creating data directory***********************************'
    mkdir -v $OUT_DIR/$PKG_NAME/data
    cp -v R_Data/*.rda $OUT_DIR/$PKG_NAME/data

    echo 'Creating documentation directory**************************'
    mkdir -v $OUT_DIR/$PKG_NAME/man
    cp -v RS_Docs/*.Rd $OUT_DIR/$PKG_NAME/man

    echo 'Creating R source code directory**************************'
    mkdir -v $OUT_DIR/$PKG_NAME/R
```

```
    cp -v R_Source/zzz.R $OUT_DIR/$PKG_NAME/R/

    # update zzz.R file; display a message that lists all of the available
    # data sets
    # SIMILAR TO SPLUS2000.bash and SPLUS6.bash
    echo 'cat("\nIncluded data sets:\n");' >> $OUT_DIR/$PKG_NAME/R/zzz.R
    echo 'cat(paste(c(' >> $OUT_DIR/$PKG_NAME/R/zzz.R
    for i in $OUT_DIR/$PKG_NAME/data/*.rda
      do echo "\"`basename $i .Rda`\"," >> $OUT_DIR/$PKG_NAME/R/zzz.R
    done
    echo 'NULL), collapse=", ")); cat("\n");}' >> $OUT_DIR/$PKG_NAME/R/zzz.R

    echo 'Listing of zzz.R*****************************'
    cat $OUT_DIR/$PKG_NAME/R/zzz.R

    echo 'Creating other file directory*****************************'
    mkdir -v $OUT_DIR/$PKG_NAME/inst
    cp -v RS_Other/COPYING $OUT_DIR/$PKG_NAME/inst/
}

# Run the R scripts to build the package
# --------------------------------------
buildPackage() {
    echo 'Building the package*****************************'

    cd $OUT_DIR
    RCmd CHECK $PKG_NAME
    if [ $? -eq 0 ]
       then
       RCmd BUILD $PKG_NAME
       RCmd BUILD --binary $PKG_NAME

       rm -vr $PKG_NAME $PKG_NAME.Rcheck
       if [ $? -ne 0 ]; then exit 1; fi
    fi
    cd ..
}

#--RUN-------------------------------------------------------------------------

if [ "$1" != "YES" ]
    then
    displayInstructions
    else
    cleanUp
    buildStructure
    buildPackage
fi
```

SPLUS2000.sh for PBSdata

```
# Cygwin's UNIX tools (sed, grep, cd, mkdir, cp, mv, ...) should be on the path.

#--CONFIGURATION---------------------------------------------------------------

# Name of package
PKG_NAME=PBSdata

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo
# Input and output directories
IN_DIR=$PROJ_ROOT/input/PBSdata
```

```
OUT_DIR=$PROJ_ROOT/output/PBSdata/S-PLUS_2000

# S-PLUS 2000's 'cmd' directory
SP2000_CMD=D:/Splus/cmd

#--FUNCTIONS-------------------------------------------------------------------

# Help display
# ------------
displayInstructions() {
    echo '
This script will create the PBS Data package for S-PLUS 2000.  If you are
producing a new version, be sure to update version numbers in the following
locations:

  * .\S_Source\zzz.S

If you have, pass "YES" as an argument to this script.'
    exit 1
}

# Remove existing packages
# ------------------------
cleanUp() {
    if [ -e $OUT_DIR ]; then
       rm -vr $OUT_DIR
       if [ $? -ne 0 ]; then exit 1; fi
    fi
}

# Build help files
# ----------------
buildHelp() {
    cd $IN_DIR

    echo 'Creating help (PBSdata.hlp)*******************************'
    mkdir -v $OUT_DIR
    mkdir -v $OUT_DIR/$PKG_NAME
    mkdir -v $OUT_DIR/$PKG_NAME/nroff

    cp -v RS_Docs/*.Rd $OUT_DIR/$PKG_NAME/nroff/
    cp -v ../utils/remusage.sh $OUT_DIR/$PKG_NAME/nroff/
    cp -v ../utils/nr2hlp.pl $OUT_DIR/$PKG_NAME/nroff/
    cp -vr ../utils/Rdconv/* $OUT_DIR/$PKG_NAME/nroff/

    cd $OUT_DIR/$PKG_NAME/nroff/

    # The script 'remusage.sh' will remove the usage section of each
    # .Rd file, since that section doesn't apply to the S-PLUS 2000 help
    bash ./remusage.sh
    rm -v remusage.sh

    # Convert *.Rd to .d
    for RdFile in *.Rd ; do
       perl Rdconv -t Sd -o `basename $RdFile .Rd`.d $RdFile
    done
    rm -v *.Rd

    # Convert *.d to .rtf
    perl nr2hlp.pl $PKG_NAME

    # Compile the .rtf file into .hlp
    hcw /c /e /m $PKG_NAME
```

```
    cd ..

    # Copy help file and clean up temporary 'nroff' directory
    mv -v nroff/$PKG_NAME.hlp PBSmapping.hlp
    rm -vrf nroff

    cd $IN_DIR
}

# Build the rest of the package
# -----------------------------
buildPackage() {
    cd $IN_DIR

    echo 'Copying S-PLUS data*********************************'
    cp -v S_Data/*.dd $OUT_DIR/$PKG_NAME/

    echo 'Copying S-PLUS source code**************************'
    cp -v S_Source/*.S $OUT_DIR/$PKG_NAME/

    echo 'Copying other files*********************************'
    cp -vrf RS_Other/* $OUT_DIR/$PKG_NAME/

    cd $OUT_DIR/$PKG_NAME/

    echo 'Creating library directory structure****************************'
    mkdir -v _Data
    mkdir -v _Data/_Help
    mkdir -v _Prefs

    # Update 'zzz.S'
    # SAME AS SPLUS6.sh
    echo 'Updating zzz.S*******************************'
    echo 'cat("\nIncluded data sets:\n");' >> zzz.S
    echo 'cat(paste(c(' >> zzz.S
    for i in *.dd
      do echo "\"`basename $i .dd`\"," >> zzz.S
    done
    echo 'NULL), collapse=", ")); cat("\n");}' >> zzz.S

    # Add commands to source data files
    for i in *.dd
      do echo "data.restore(\"$i\");" >> zzz.S
    done

    echo 'Listing of zzz.S*****************************'
    cat zzz.S
    # /SAME AS SPLUS6.sh

    # Prepare 'script'
    echo 'source ("zzz.S");' > script

    # Start S-PLUS in batch mode using 'script'
    echo 'Building library*******************************'
    $SP2000_CMD/splus /BATCH script S_PROJ=.

    # Finish cleaning up
    echo 'Cleaning up*********************************'
    rm -vf zzz.S
    rm -vf *.dd script tmp*
    rm -vrf _Prefs/*
```

```
    cd ..

    zip -r -9 $PKG_NAME $PKG_NAME
    rm -vr $PKG_NAME
}

#--RUN-------------------------------------------------------------------------
if [ "$1" != "YES" ]
    then
    displayInstructions
    else
    cleanUp
    buildHelp
    buildPackage
fi
```

SPLUS6.sh for PBSdata

```
# Cygwin's UNIX tools (sed, grep, cd, mkdir, cp, mv, ...) should be on the path.

#--CONFIGURATION---------------------------------------------------------------

# Name of package
PKG_NAME=PBSdata

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo
# Input and output directories
IN_DIR=$PROJ_ROOT/input/PBSdata
OUT_DIR=$PROJ_ROOT/output/PBSdata/S-PLUS_6

# R root directory
R_ROOT=D:/R/rw1091

# Cygwin bin directory
CYG_BIN=D:/Utils/Cygwin/bin

# Microsoft HTML Help Workshop directory
HHW_DIR=D:/Utils/HTML-Help-Workshop

# S-PLUS 6.x's CHM tools directory
CHM_TOOLS_DIR=D:/Splus62/help/BuildHelpFiles

#-------------------------------------------------------------------
# Variables for running the S-PLUS 6 'build_helpfiles' script
# See build_helpchm.cyg in $CHM_TOOLS_DIR
#-------------------------------------------------------------------
CHM_DIR=$OUT_DIR/$PKG_NAME  # Path to CHM file (that will be created)
CHM_NAME=$PKG_NAME          # Basename of the CHM file
HEADER="Language Reference for Library PBS Mapping"
DEF_PAGE=SOM.html           # Default page when help first loaded
SEARCH_STOPIC_LIST=T        # S-topic list to link to splus.chm
STOPIC_LIST_SPLUS=$CHM_TOOLS_DIR/stopicList.out
#GUI_HELP_DIR=$CHM_DIR/html  # If including HTML files
HAVE_HTML_FILES=F
SGML_DIR=$CHM_DIR/sgml      # Path to SGML files
HTML_DIR=$SGML_DIR          # Path to root of HTML files


#--FUNCTIONS-------------------------------------------------------------------

# Help display
```

```
# ------------
displayInstructions() {
    echo '
This script will create the PBS Data package for S-PLUS 6.
If you are producing a new version, be sure to update version numbers
in the following locations:

  * .\S_Source\zzz.S

If you have, pass "YES" as an argument to this script.'
    exit 1
}

# Remove existing packages
# ------------------------
cleanUp() {
    if [ -e $OUT_DIR ]; then
       rm -vr $OUT_DIR
       if [ $? -ne 0 ]; then exit 1; fi
    fi
}

# Build help files
# ----------------
buildHelp() {
    cd $IN_DIR

    echo 'Creating help (PBSmapping.chm)*******************************'
    mkdir -v $OUT_DIR
    mkdir -v $OUT_DIR/$PKG_NAME
    mkdir -v $OUT_DIR/$PKG_NAME/sgml

    cp -v RS_Docs/*.Rd $OUT_DIR/$PKG_NAME/sgml/
    cp -v ../utils/remusage.sh $OUT_DIR/$PKG_NAME/sgml/
    cp -v S_Other/SOM.sgml $OUT_DIR/$PKG_NAME/sgml/
    cp -vr ../utils/Rdconv/* $OUT_DIR/$PKG_NAME/sgml/

    cd $OUT_DIR/$PKG_NAME/sgml/
    # The script 'remusage.sh' will remove the usage section of each
    # .Rd file, since that section doesn't apply to the S-PLUS 6 help
    bash ./remusage.sh
    rm -vf remusage.sh

    # Convert all the .Rd files to .sgml
    for RdFile in *.Rd ; do
       perl Rdconv -t Ssgm -o `basename $RdFile .Rd`.sgml $RdFile
    done
    rm -v *.Rd

    # Call the S-PLUS utility that will build the help files
    $CHM_TOOLS_DIR/build_helpfiles $CHM_NAME $SGML_DIR $HTML_DIR $CHM_TOOLS_DIR
$HHW_DIR $DEF_PAGE $CHM_DIR $CYG_BIN "$HEADER" $SEARCH_STOPIC_LIST $STOPIC_LIST_SPLUS
$GUI_HELP_DIR $HAVE_HTM_FILES

    # Check help.log for errors
    cd ..
    nerr=$(grep -c -i -w "error" help.log)
    echo $nerr
    if [ $nerr -gt 0 ]; then
       echo Terminating: Check help.log \in `pwd | cygpath -ms -f -`
       exit 1
    fi
```

```
    mv help.log $OUT_DIR/
    rm -rf sgml
    rm -rf deadlinks_postfix.lst deadlinks_prefix.lst help.log

    cd $IN_DIR
}

# Build the rest of the package
# -----------------------------
buildPackage() {
    cd $IN_DIR

    echo 'Copying S-PLUS data*******************************'
    cp -v S_Data/*.dd $OUT_DIR/$PKG_NAME/

    echo 'Copying S-PLUS source code*******************************'
    cp -v S_Source/*.S $OUT_DIR/$PKG_NAME/

    echo 'Copying other files*******************************'
    cp -vrf RS_Other/* $OUT_DIR/$PKG_NAME/

    cd $OUT_DIR/$PKG_NAME/

    # Update 'zzz.S'
    # SAME AS SPLUS2000.sh
    echo 'Updating zzz.S*******************************'
    echo 'cat("\nIncluded data sets:\n");' >> zzz.S
    echo 'cat(paste(c(' >> zzz.S
    for i in *.dd
      do echo "\"`basename $i .dd`\"," >> zzz.S
    done
    echo 'NULL), collapse=", ")); cat("\n");}' >> zzz.S

    # Add commands to source data files
    for i in *.dd
      do echo "data.restore(\"$i\");" >> zzz.S
    done

    echo 'Listing of zzz.S*******************************'
    cat zzz.S
    # /SAME AS SPLUS2000.sh

    # Call the S-PLUS 'CHAPTER' command, which will source all of
    # the *.S files and build the package.
    echo 'Building library*******************************'
    CHAPTER -s

    echo 'Cleaning up*******************************'
    rm -v zzz.S
    rm -vr *.dd .Data/__Hhelp

    cd ..

    zip -r -9 $PKG_NAME $PKG_NAME
    rm -vr $PKG_NAME
}

#--RUN---------------- ------------------------------------------------------------
if [ "$1" != "YES" ]
    then
    displayInstructions
    else
    cleanUp
```

```
    buildHelp
    buildDLL
    buildPackage
fi
```

PDF.sh for PBSmapping and PBSdata

```
# ********** Configuration **********

# Project directory root, with subdirectories ./input and ./output
PROJ_ROOT=E:/Archive/Projects/Mapping/Devo

# R root directory
R_ROOT=D:/R/rw1091

# MikTeX bin directory
MIK_BIN=D:/Utils/MikTex/MikTex/bin

# ********** Set variable names **********

# Package name for this appendix
PKG_NAME=$1

# Input and output directories
INPUT_DIR=$PROJ_ROOT/input/appendix
OUTPUT_DIR=$PROJ_ROOT/output/appendix

# ********** Define functions **********

# Debug
debugScr() {
  echo $PROJ_ROOT
  echo $R_ROOT
  echo $MIK_BIN
  echo $PKG_NAME
  echo $INPUT_DIR
  echo $OUTPUT_DIR
}

# Error display
displayInstructions() {
    echo 'Supply script argument PBSmapping or PBSdata'
    exit 1
}

# Clean output directory
cleanUp() {
   echo '********** Cleaning up existing output directory...'
    if [ -e $OUTPUT_DIR/$PKG_NAME ]; then
       rm -vr $OUTPUT_DIR/$PKG_NAME
       if [ $? -ne 0 ]; then exit 1; fi
    fi
}

prepareDocs() {
    cd $INPUT_DIR
    echo '********** Creating and output directories'
    mkdir -v $OUTPUT_DIR
    mkdir -v $OUTPUT_DIR/$PKG_NAME
    mkdir -v $OUTPUT_DIR/$PKG_NAME/tmp
    cp -vf tex/App$PKG_NAME.tex $OUTPUT_DIR/$PKG_NAME/tmp/
    cp -vf ../$PKG_NAME/RS_Docs/*.Rd $OUTPUT_DIR/$PKG_NAME/tmp/
```

```
    cp -vf ../utils/remusage.sh $OUTPUT_DIR/$PKG_NAME/tmp/
    cp -vf ../utils/Rd2tex.sh $OUTPUT_DIR/$PKG_NAME/tmp/
    cp -vf $R_ROOT/share/texmf/*.* $OUTPUT_DIR/$PKG_NAME/tmp/

    # Remove the "\usage{data" section from each Rd file about a data object.
    # This appears redundant in appendix listings.
    echo '********** Removing "Usage" from Rd files about data objects'
    cd $OUTPUT_DIR/$PKG_NAME/tmp
    bash ./remusage.sh
    cd $INPUT_DIR
}

buildPackage() {
    cd $OUTPUT_DIR/$PKG_NAME/tmp
    echo '********** Converting *.Rd to .tex...'
    for i in *.Rd
      do $R_ROOT/bin/rcmd rdconv -t latex -o `basename $i .Rd`.tex $i
    done

    # Depending on the package, the sort method changes.
    # For PBS Mapping, ignore the case while sorting. The functions/files
    # that contain upper case should appear with the rest.
    # For PBS Data, perform a "dictionary" sort.  The advantage with this sort
    # is that 'ltsa.dat' will follow 'ltsa' (and the same for 'pfma')
    if [ $PKG_NAME == "PBSmapping" ]; then
       SORT='sort -f'
    else
       SORT='sort -d'
    fi

    # Add all input file names to the file App$PKG_NAME.tex. By building
    # this list at run-time, this TeX template won't need to be updated
    # with the addition of new files
    ls -1 *.Rd\
       | $SORT\
       | sed s/.Rd/.tex/g\
       | sed s/^/\\\\input{/g\
       | sed s/$/}/g\
       >> App$PKG_NAME.tex
    echo '\end{document}' >> App$PKG_NAME.tex

    echo '********** Creating PDF file...'
    $MIK_BIN/pdflatex App$PKG_NAME.tex

    mv -v App$PKG_NAME.pdf ../
    mv -v *.tex ../
    cd ..
    rm -vr tmp
}

debugScr
if ( [ $PKG_NAME != "PBSmapping" ] && [ $PKG_NAME != "PBSdata" ] ); then
    displayInstructions
else
    cleanUp
    prepareDocs
    buildPackage
fi
```